32nd IEEE Conference on Local Computer Networks

Closing the Loop for Dynamic IP QoS Provisioning: A Case Study

Swati Sinha Deb, Alistair Munro
Department of Electrical and Electronic Engineering
University of Bristol
Bristol, UK
Email: {Swati.Deb, Alistair.Munro}@bristol.ac.uk

***Abstract*—This paper investigates an approach to enabling dynamic reactive quality-of-service (QoS) management for IP networks. We use the information collected from monitoring traffic flows combined with middleware for setting network element configurations to differentiate the service given to traffic. We report on experiences of implementing the QoS management system using XORP (eXtensible Open Router Platform) and Click, the open source software router. The performance evaluation in terms of packet loss, dynamic behaviour of QoS configuration and time to re-configure the XORP+Click router is presented.**

I. Introduction

There is a class of applications, such as performance management (for traffic monitoring, configuration tuning and planning purposes), usage-based accounting, or attack/intrusion detection, that require traffic measurements from the network. They deliver information to which the provider can be expected to react over a relatively long timescale. By contrast, there is a collection of issues of interest concerning the use of traffic measurement to discover what is actually happening in an IP network, and subsequently to control closer to real-time how packets are processed, i.e. closing control loop without operator intervention. This is topical for "the Internet", i.e. the collection of internetworked communications systems running IP protocols. The Internet thrives on diversity and openness but this has a potential downside when exploited for malicious purposes or actively attacked. While policing such undesirable activity is probably a good thing to do, it does require mechanisms that are themselves as incorruptible as necessary while being simple, having low cost (in additional traffic load or system complexity, or operator expertise), and being configurable and mobile (i.e. they can be put where they are useful and usable).

In this paper we present a software prototype implementation that integrates simple off-the-shelf components to build flexible passive measurement system that has the essence of these attributes. In summary, the proposed measurement system consists of an IPFIX (IP Flow Information Export) compliant meter (as in [1]), exporter and collector. The flows are detected at line speed using a Tarari regular expression (regex) agent [2] [1]. We use a relational database to query and store the measurement data for further analysis by various applications to troubleshoot and answer various typical questions related to network performance. To demonstrate the proposed network measurement system we consider QoS monitoring as a use case. The deployment of QoS provisioning within an Internet has generated lot of interest recently within the research community, (but not very much in the user and operator domain). One way to deploy QoS on the Internet is through the IETF (Internet Engineering Task Force) defined policy-based routing. Our prototype uses this in combination with the differentiated service (DiffServ) model as a demonstrator of QoS management applicable to IP networks. It is reminiscent in some ways of the UMTS QoS architecture specified by 3GPP, although we are aiming at a bi-directional reactive system in which QoS management is just one possible supported function.

The paper is organized as follows. In section II we describe the flow-export protocol used in the prototype, and discuss why we chose it in preference to the current IPFIX protocol. Section III describes the software implementation of the network monitoring system. Section IV demonstrates the testbed and performance of the software-based edge router architecture in which the forwarding plane of the router is dynamically configured to enable QoS based on the network policies, transport, middleware and application functions. It also illustrates performance evaluation of the router in terms of dynamic behaviour of router, throughput and time to reconfigure the router. Section V of the paper discusses existing work. Finally, section VI concludes the paper and identifies future work.

II. Traffic Measurement: Processes and Protocols

The two techniques used to measure network traffic are active measurement and passive measurement. Active measurement sends probe packets into the network. Tools like ping and traceroute are fairly crude examples of this; other better conceived tools have been proposed by IETF working groups. Other techniques are indirectly active, such as the

[1] Some terms used in this paper may be registered trademarks of corporations including Tarari, XACCT, and Cisco. Where used, we intend them to be interpreted in the same way as they are understood in common usage in the IP community.

0742-1303/07 $25.00 © 2007 IEEE
DOI 10.1109/LCN.2007.56

round-trip time, or delay, estimates used in some multicast congestion management models (PLM, WEBRC, PSLM). The active approach creates extra traffic and the traffic, or its parameters, are artificial. By contrast, passive measurement utilizes the existing traffic in the network to gather traffic statistics. One way of doing this is to poll passive monitoring devices periodically and record responses that are used to assess network performance and status. It is typically applied where the traffic of interest is already present. Passive measurement can be performed using sniffers or OC3MON [3], or it can be embedded in routers, switches or end hosts. Remote Monitoring (RMON) [4], which is based on IP network management protocol, (SNMP), Cisco's NetFlow [5] and Intel's CoMo (Continuous Monitoring) [6] are some examples of embedded functions.

Intel's CoMo shows typical features that could be expected of a modern passive measurement system:

- an open software platform;
- allows any generic metric to be computed on an incoming traffic stream;
- provides privacy and security guarantees to the network users, CoMo users and the owner of the monitoring link;
- it is robust in a sense that no "black-out" periods occur when it cannot sustain the incoming traffic streams;
- can be extended and personalised using plug-in modules, which are responsible for various transformations of the data;
- export to external mass storage.

CoMo's core processes perform basic operations such as packet capture, filter and data storage. Incoming packets are processed in various ways, according to configuration, and finally stored onto hard disks from where the data are retrieved on user request. CoMo uses a stream database approach to manage network data queries.

The communications community has produced various proposals for, or adopted de-facto solutions as, standards. For example, at protocol level, the IETF IPFIX working group [1] has defined how to export IP flow measurement data collected at different points at the network from IPFIX exporter to IPFIX collectors. At present IPFIX is based on Cisco's NetFlow to export IP measurement data to different range of management applications. Groups interested in charging have produced their own proposals.

A. Choosing a Flow-Export Protocol

The IPFIX working group assessed several export protocols and associated architectures, and chose a model based on Cisco's NetFlow. We chose CRANE (Common Reliable Accounting for Network Element) protocol [7] because we believe it has its own strengths, and the differences and similarities deserve further exploration in any case. We aim in the long term to address the following questions, motivated by:

1) Origin and history - the CRANE protocol emphasizes extensibility with the goal of applicability to a wide range of accounting tasks and business processes, i.e. it is more than a traffic or packet capture and export system. What benefits and new insights could this perspective bring?
2) Issues of methodology, architecture, information flow, and function - these provide a cultural framework that underlies the protocol. Is the CRANE protocol a richer language that can be used to build more powerful flow export and management systems?
3) Sampling granularity - what can be ignored, or even that nothing is ignored. What loss is tolerable?
4) Transport of data - the usual choice of unreliable vs. reliable underlying services, and their coexistence with other services and applications sharing the network. What recommendation is best?
5) Security - is a flow export system more of a threat to overall security and availability than an asset to the network?
6) Event notifications - how to select events and who reports them?
7) High-availability and resilience - does the language spoken between exporters and collectors make it easier to achieve a high grade-of-service?

Fig. 1. NetFlow Version 9 packet layout

The main characteristics Cisco's NetFlow and CRANE are given below, for the purposes of comparison, and the reader should please refer to [8] for a detailed analysis of other existing protocols such as DIAMETER, LFAF, and Streaming IPDR. Note: we refer to NetFlow version 9 when we write NetFlow.

- Both CRANE and NetFlow provide a mechanism for defining the layout (ordering and size of fields) and semantics (datatype) of exported data records using a template approach. A NetFlow exporter will send the template definitions to the collector before sending flow records. One of the key elements in NetFlow is the template FlowSet which describes the type and length of individual fields within following NetFlow data records that match a template type. NetFlow also provides for an option template and its corresponding options data record. The option template set is used to provide information regarding "meta-data" about the NetFlow process itself and not about IP flows. The NetFlow record format, as shown in Fig. 1, consists of packet header followed by atleast one or more template or data FlowSets. Fig. 2 shows CRANE packet. In a CRANE system there is a phase of template negotiation between server(s) and a client participating in a given session before actual accounting data is sent. CRANE uses "keys" to specify the fields in the template. A key can be enabled or disabled. An enabled key means that the data record

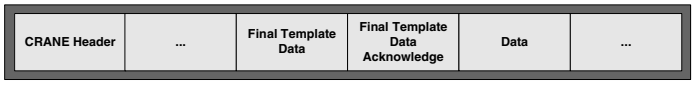

Fig. 2. CRANE packet for a given CRANE session

will contain the data item specified by the key whereas a disabled key will skip the specified data item. The enabling and disabling of keys in template negotiation affects the bandwidth requirement. After the template negotiation, the CRANE client only needs to send actual accounting data without any descriptors of the data. This reduces the processing in network elements.

- In NetFlow the data is represented using network byte order whereas the CRANE protocol can choose the byte ordering. The key advantage is that this lowers the processing demand on exporter based on little-endian architectures.
- In NetFlow the data flow is unidirectional. The data flow is basically from exporter to collector, with the collector only sending acknowledgements. In CRANE protocol the flow of control messages between the exporter and collector are bi-directional.
- NetFlow has been designed to be transport protocol independent. Hence, it can operate over congestion-aware protocols such as SCTP, as well as TCP and UDP. CRANE protocol operates over either SCTP or TCP transport protocol, and inherits the congestion awareness of these protocols. In NetFlow, the use of "reliable" transport protocol is optional, and it does not support application-level acknowledgements. CRANE uses application-layer acknowledgements as an indication of successful processing by the CRANE server.
- CRANE protocol can either use lower-layer security mechanisms such as IPSEC or TLS whereas NetFlow over UDP relies on IPSEC. NetFlow does not impose any confidentiality, integrity or authentication requirements as this reduces the efficiency of the implementation.
- CRANE supports redundant server configurations and fault tolerance. NetFlow allows currently two collectors to be configured in an exporter. Both the collectors in NetFlow receive all the data records and could use the sequence number or inter-collector communication to de-duplicate the data records. This may waste bandwidth and other network resources.
- NetFlow supports sampling. CRANE considers sampling to be inadmissible as it is targeted towards Telco-grade accounting.

In brief the key attributes of the CRANE protocol are:

End-to-end reliability: it uses a transport layer protocol such as TCP or SCTP that ensures in-sequence, reliable data packet delivery. It is bi-directional over the entire duration of a session. It implements protocol level acknowledgements that ensure the accounting records are stored and processed successfully in the CRANE server(s).

Efficiency: it is the only protocol that can choose the byte ordering of the data. Using "template negotiation" only desired data records and fields are transmitted with considerably less overhead compared to other data encoding techniques.

Flexibility: it imposes minimal constraints on the data structure of transmitted accounting records. Implementing a "template negotiation" process can support any user-defined data structure. The protocol also has version and capability negotiation facility.

Scalability: it scales well with an increasing number of network elements due to its high performance, low overhead and load balancing capability.

III. Network Traffic Monitoring System

A. Software Implementation

The prototype consists of an IPFIX meter, exporter and collector. The other components that build the complete measurement system are traffic generator, XORP (eXtensible Open Router Platform) [9] + Click modular software router [10], Tarari hardware regex agent that acts as a filter, CRANE protocol to export the metered data records to collector and MySql database to store the data records. The complete system is shown in Fig. 3.

The packet classification function is the most frequently exercised part of the system, and a potential bottleneck, so we use the Tarari regex agent because of its powerful firmware capabilities [2]:

1) High speed pipelined architecture
   - "DFA" style state machine for maximum speed
   - Processes five jobs simultaneously for throughput of up to 1 Gbps per regex agent
   - Support for load balancing across 4 regex agents for either maximum regular expression capacity of 4 Gbps throughput
2) Two pattern matching modes
   - Parsing mode in which a prioritized greedy matching strategy uniquely labels input byte sequences as lexemes
   - Pattern Matching mode in which every pattern that matches is reported regardless of overlap
3) Multiple selectable outputs formats
   - Token list with start and end pointers
   - Token list with just start pointers
   - Token list with no pointers
   - Bit vector
4) Tarari regex analysis tool
   - Reports a Pareto analysis of state usage per regex
   - Enables targeted optimization of the most resource intensive expressions

The measurement application runs as a process in user-space. The IPFIX meter and exporter consists of a Linux/PC machine. A traffic generator supplies packets from different applications and these are captured in pcap format at one of the configured interfaces of XORP + Click software router. This is known as the observation point. Timestamping is performed using

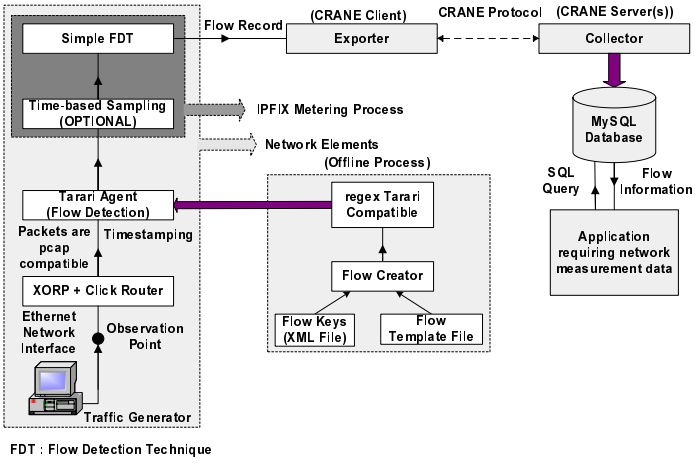

Fig. 3. Network measurement system architecture

```
<?xml version="1.0"?>
<!DOCTYPE FLOWDATA SYSTEM "file://h:\XML\FlowData.dtd">

<FLOWDATA>
        <FLOW FLOWNAME="DEFAULT">
                <FLOWKEY NAME="MacDstAddr" VALUE="{ANYBYTE}{6}"/>
                <FLOWKEY NAME="MacSrcAddr" VALUE="{ANYBYTE}{6}"/>
                <FLOWKEY NAME="Type" VALUE="\x08\x00"/>
                <FLOWKEY NAME="IPProto" VALUE="6"/>
                <FLOWKEY NAME="IPSrcAddr" VALUE="{ANYBYTE}{4}"/>
                <FLOWKEY NAME="IPDstAddr" VALUE="{ANYBYTE}{4}"/>
                <FLOWKEY NAME="TCPSrcPort" VALUE="2000"/>
                <FLOWKEY NAME="TCPDstPort" VALUE="80"/>
        </FLOW>
        <FLOW FLOWNAME="FLOW1">
                <FLOWKEY NAME="MacDstAddr" VALUE=" "/>
                <FLOWKEY NAME="MacSrcAddr" VALUE=" "/>
                <FLOWKEY NAME="Type" VALUE=" "/>
                <FLOWKEY NAME="IPProto" VALUE="6"/>
                <FLOWKEY NAME="IPSrcAddr" VALUE="172.20.1.2"/>
                <FLOWKEY NAME="IPDstAddr" VALUE="172.20.3.3"/>
                <FLOWKEY NAME="TCPSrcPort" VALUE="2000"/>
                <FLOWKEY NAME="TCPDstPort" VALUE="80"/>
        </FLOW>
</FLOWDATA>
```

Fig. 4. Flow key xml file

the respective Click element, which records in seconds and microseconds when a packet is received.

B. System Architecture

The overall architecture can be divided into an offline component where the user specifies the flow definition and a near real-time component which is the IPFIX metering process. The main building blocks of metering process are packet header capturing, timestamping, packet filtering, sampling and classifying. The key task of the metering process is to generate flow records.

The offline component, i.e., the specification of the flow consists of two parts. The first is the flow keys XML file, as shown in Fig. 4, containing flow keys corresponding to a particular flow definition. A flow is defined as a set of IP packets passing an observation point (i.e. a defined interface into a router) in the network during a certain time interval. In the current implementation, all packets belonging to a particular flow have a set of common properties such as source

```
%option warn
ANYBYTE             .
IP_PKT              {ANYBYTE}{9}
TCP_PROTO           \x06
UDP_PROTO           \x11
IP_HDR_CHKSUM       {ANYBYTE}{2}
IP_SRC_ADDR1        \xac\x14\x01\x02
IP_DST_ADDR1        \xac\x14\x03\x03
IPHEADER1           {IP_PKT}{TCP_PROTO}{IP_HDR_CHKSUM}{IP_SRC_ADDR1}{IP_DST_ADDR1}
IPHEADER2           {IP_PKT}{UDP_PROTO}{IP_HDR_CHKSUM}{IP_SRC_ADDR1}{IP_DST_ADDR1}
TCP_SRC_PORT1       \x07\xD0
TCP_DST_PORT1       \x00\x50
TCP_SRC_PORT2       \x1f\x40
TCP_DST_PORT2       \x23\x29
TCP_FRAME           {ANYBYTE}{16}
TCPHEADER1          {TCP_SRC_PORT1}{TCP_DST_PORT1}{TCP_FRAME}
TCPHEADER2          {TCP_SRC_PORT2}{TCP_DST_PORT2}{TCP_FRAME}
UDP_SRC_PORT1       \x17\xa7
UDP_DST_PORT1       \x0f\xd7
UDP_FRAME           {ANYBYTE}{4}
UDPHEADER1          {UDP_SRC_PORT1}{UDP_DST_PORT1}{UDP_FRAME}


%s      IPHDR IPTCPHDR IPUDPHDR
%%

{IPHEADER1}             {
                                << NO OUTPUT >>
                                << START IPTCPHDR >>
                        }
{IPHEADER2}             {
                                << NO OUTPUT >>
                                << START IPUDPHDR >>
                        }
<IPTCPHDR>{TCPHEADER1} {
                                << OUTPUT 1 >>
                                << HALT >>
                        }
<IPTCPHDR>{TCPHEADER2} {
                                << OUTPUT 2 >>
                                << HALT >>
                        }
<IPUDPHDR>{UDPHEADER1}{
                                << OUTPUT 3 >>
                                << HALT >>
                        }
```

Fig. 5. Tarari regex template

IP address, destination IP address, network protocol (IP, ICMP, etc.), application protocol (which can be related to the port number in many cases provided conventions are followed). A single flow key file can have multiple flow definitions. Each flow key is a name/value pair. The second component in the offline process is the flow template file. The template file is of similar structure to Tarari regex specification. The template file represents the regular expression to identify complex network flows. The template file is designed with replaceable parameters where the parameter values can be picked up from the flow key XML file (name/value pairs). The process of merging the XML based flow key file and flow template file will be performed offline by a flow creator, a Perl based parser. The output of this parser will be a regex file, as shown in Fig. 5, which will contain regular expressions to identify flows. This regex file is then used by the Tarari API to perform pattern matching. Tarari agent acts as a hardware filter that sort packets of a flow from incoming streams of flows according to the selection criteria specified in the flow keys XML file. The Tarari infrastructure identifies a flow and reports the results to the next process. Optionally, one can have a software filter instead of hardware filter.

The next process is sampling. Sampling refers to selecting a subset of the traffic captured at observation point. Sampling techniques can be classified as random sampling, where packet selection is based on random functions (such as n-out-of-N, uniform probabilistic, non-uniform probabilistic) and systematic sampling, based on deterministic functions. The selection

decision in systematic sampling can depend on packet timestamp (time-based), packet count (count-based) or packet content (content-based). Content-based sampling includes hash-based methods. We have a provision of time-based sampling and this represents a deviation from the CRANE model. Packet selection is triggered at periodic instants separated by a time called the spacing. All packets that arrive within a certain time of the trigger (called the interval length) are selected. At present we are using a very simple technique to create flow records from input stream of packet or session records. These are formatted as single line records comprising standard fixed fields, such as timestamp, IP addresses, ports, that are reported for every record and other data components reported using the widely favoured Type-Length-Value (TLV) format. We use the flow 5-tuple consisting of source IP address, destination IP address, transport protocol, source port and destination port as a hash to update and associate the records in the Flow Record array.

The exporter is an entity that performs the transfer of measurement data to the collector. It accesses the flow record cache, selects the Flow Records to be exported and builds CRANE messages out of them.

The collector is an entity that receives and stores the measurement data from an exporter.

The final component is a relational database where the collector stores the measurement data. We chose to use MySql database for the following reasons:

- It should be fairly easy for the server to insert new data.
- It should be easy to retrieve data when a query is submitted from the management applications.

1) CRANE Protocol Software: The software implementation of the CRANE protocol is shown in Fig. 6, to transport the metered data from upstream systems such as network elements to downstream systems. The CRANE protocol defines the necessary messages that need to be exchanged, such as connect, flow start, template data, error, flow stop. Each message is composed of specific CRANE objects.

The CRANE client is implemented on the exporting side (data producing side) and the CRANE server is implemented on the collecting side (data receiving side). The CRANE client is integrated with the network element's software, enabling it to collect and send the usage information to the CRANE server

In CRANE client there are two threads running. The multi-threading is implemented using the ACE (Adaptive Communication Environment) Framework [11]. One of the processes in the CRANE client, called the Flow Record Manager, reads the flow records from input stream and stores locally into the Flow Record Cache. The Flow Record Cache is a set of structure of different types of data record. Once the Flow Record Cache is full, the oldest records are dropped first to accommodate the new flow records. The CRANE client will retrieve the flow records from the Flow Record Cache.

The CRANE protocol is an application running over a reliable transport layer such as TCP or SCTP. In our design, the CRANE messages are transmitted using TCP. The CRANE server connects to the CRANE client, and is responsible for re-establishing a connection in case of a failure. Another process in the CRANE client spawns thread to accommodate connections from multiple servers simultaneously. Once a CRANE server has established a connection with the CRANE client; the CRANE client then configures the network address of all the CRANE servers belonging to the session. It also maintains the CRANE server's status which reflects whether the server is active or inactive and the priority of the server. The transport layer is responsible for monitoring the CRANE server's responsiveness and notifying the CRANE client of any failures. The server priority reflects the CRANE client's preference regarding which CRANE server should receive the accounting data.

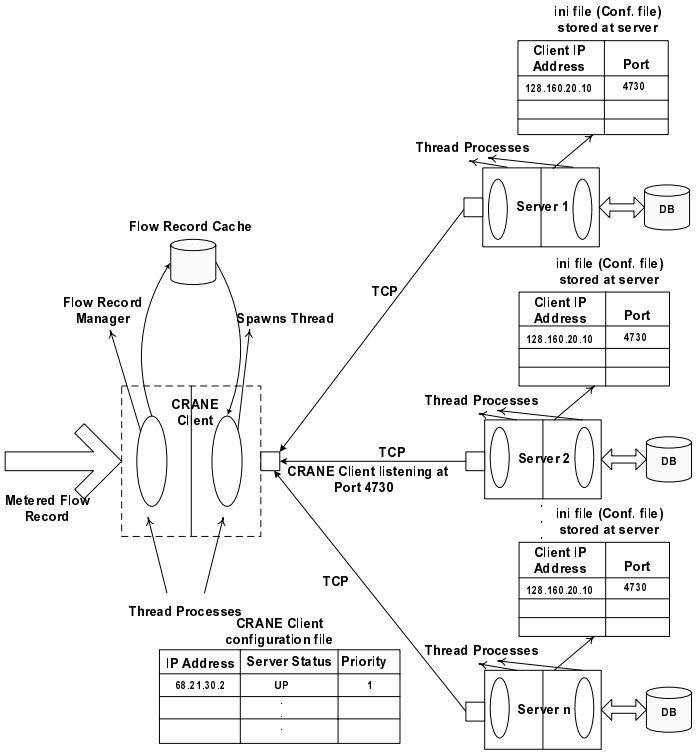

Fig. 6. Software implementation of CRANE protocol

Another thread in the CRANE server(s) stores the received accounting data into the database.

IV. Application Case Study -QoS Management

Fig. 7 shows the testbed used for the QoS management experiment. The testbed is composed of traffic transmitter, traffic receiver and XORP+Click routers that are connected by 10Mbps Ethernet links. It has a policy client and CRANE client installed. The CRANE client is used to transfer the network measurement data to the CRANE server. The CRANE server then stores the flow information into the SQL database, which can be used by various management applications such as attack/intrusion detection, traffic engineering for further analysis, and, in our case, QoS management.

The Bandwidth Manager sends an SQL query to retrieve the flow properties. It then determines the rate of the session

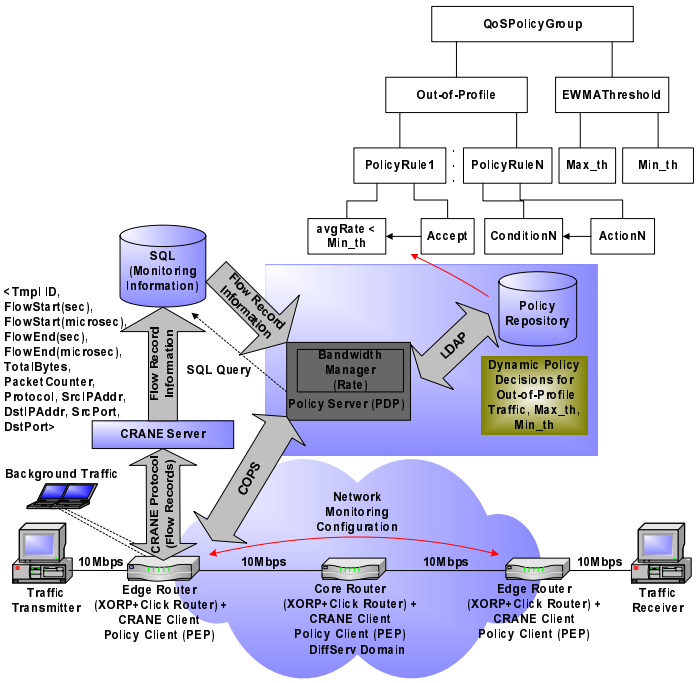

Fig. 7. QoS management testbed

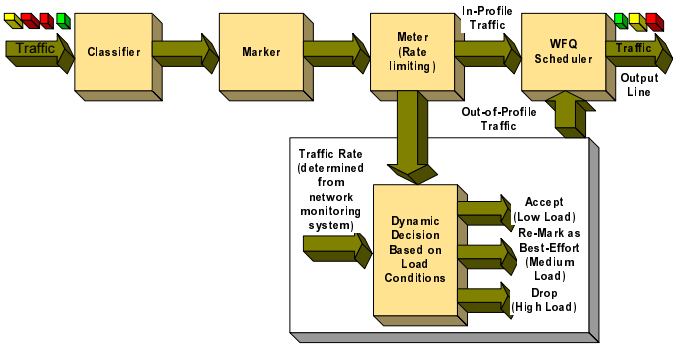

Fig. 8. Dynamic QoS decision

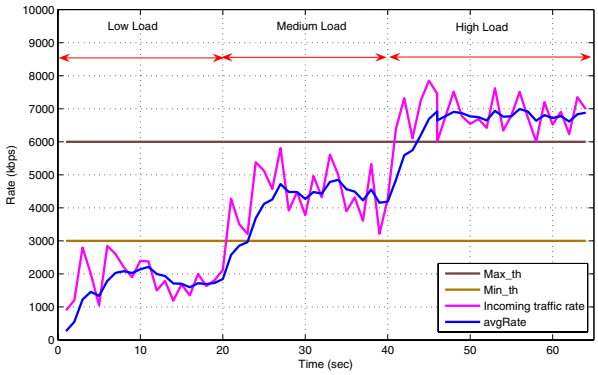

Fig. 9. Input traffic: dynamic behaviour of XORP+Click router

(or flow). Using this rate Bandwidth Manager calculates the $avgRate$, which is achieved by using the exponential weighted moving average (EWMA) [12]. It then communicates this to the policy server. The policy server makes a decision based on the policy rules it retrieves from the policy repository, which is an openLDAP (Lightweight Directory Access Protocol) [13] server. Our implementation uses the three tier policy model as described in [14] to store the policies.

The internal router configuration of forwarding plane, such as classification, marker, meter and scheduler are Click elements, are dynamically adjusted to guarantee each traffic flow QoS requirements. Based on the $avgRate$ and the thresholds, which is used to determine the load condition of the network, the policy server makes a dynamic decision of which policy to use for out-of-profile traffic, i.e., to accept, remark as best effort or drop, as illustrated in Fig. 8.

The policy server then sends the decision to the policy client using Common Open Policy Service (COPS) protocol [15]. Each time the policy condition changes the XORP interfaces are re-configured according to the new Click configuration file. The new Click configuration file is installed by the policy client using the user-level hot-swapping option. The hot-swapping in the user-level is performed using the Click ControlSocket element, which opens a control socket that allows other application to call read/write handlers on the router.

A. Performance Analysis

The experiment performed aims at estimating the efficiency and responsiveness of the system.

The first experiment focuses on dynamic configuration of the router. The traffic generator sends UDP packets of 256 bytes. We send 400-4000 packets per sec to load the network differently each time. The test is run for 60 seconds and every 20 seconds we change the load, i.e., from low load to medium load and then high load. To achieve low load conditions the traffic generator sends 400-1400 packets per second randomly, for medium load it sends 1500-2900 packets per second and for high load it sends 3000-4000 packets per second. Policing is performed at the edge of the network for a given traffic, which is identified by flow 5-tuple as described above. Policy is determined by the traffic specification. In this experiment we set the meter rate limit to 1000kbps, i.e. the incoming traffic must not exceed 1000kbps. For testing purposes we transmit out-of-profile traffic (traffic that has rate equal or greater than 1000kbps). The out-of-profile traffic can be accepted, re-marked as best effort or dropped depending on the load conditions. The in-profile traffic is that conforms to the traffic specification. Following DiffServ protocol, the in-profile traffic is marked with IP Precedence of 6 to give higher priority.

Fig. 9 shows the link load during the period of the experiment. This measurement was taken at the ingress interface of the edge router. We set the Max_th to be 6000kbps and Min_th to be 3000kbps. The Bandwidth Manager retrieves the flow properties from the monitoring system every 1 second to determine the rate. The policy server makes the decision according to the $avgRate$.

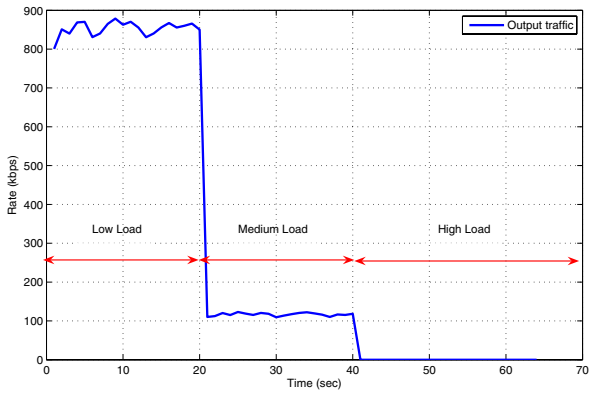

Fig. 10. Outgoing traffic: dynamic behaviour of XORP+Click router

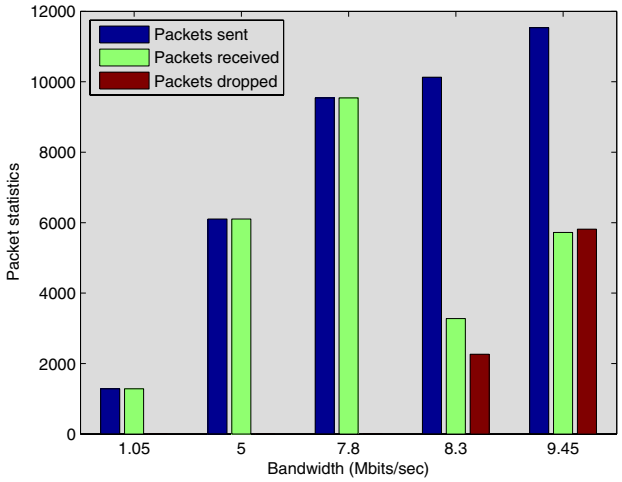

Fig. 11. Packet loss

Fig. 10 shows received traffic behaviour. The out-of-profile traffic is dynamically handled depending upon the load conditions. When the load is medium, a high priority packet is remarked as best effort packet and then shaped at 125kbps and in low load condition the out-profile traffic is accepted and shaped at 875kbps. Similarly, when the load is high all the packets are dropped.

The average time to re-configure the router for the simple policy rules used in the experiment is approximately 45 μsec.

The next experiment determines the throughput using the IPerf tool [16] for evaluating packet loss. It generates a single UDP flow and runs for 10 seconds. The results depicted in Fig. 11 show no packet losses for throughput below 7.8Mbps or equivalently 952 packets per second (packets are 1024 byte datagrams). Beyond that threshold the losses are due to the Click element processing limitations inside the router.

Table I shows the execution times (in microseconds per packet) of each element involved in providing QoS in software router. The result represent a low level of performance because the implementation is in user space. We use 64 byte UDP packets as minimum size packets stress the router harder than larger packets; the CPU and several other bottleneck resources are consumed in proportion to the number of packets forwarded, not in proportion to bandwidth.

Element	Time(μsec)
Strip	36
CheckIPHeader	65
SetTimestamp	25
Meter	23
TarariHandler	280
FlowClassifier	33
Shaper	57
WFQSched	112
LinearIPLookup	34
RED	60

TABLE I
EXECUTION TIME REQUIRED BY EACH ELEMENT TO ENABLE QOS IN XORP + CLICK SOFTWARE ROUTER, IN MICROSECONDS PER PACKET

V. RELATED WORK

Traditional sources of traffic monitoring are SNMP, Cisco's NetFlow and packet level data. SNMP provides information about the management data which describes the system configuration. SNMP can be used to determine the volume of traffic (in bytes), monitoring device uptimes, and network interface throughput. Flow-level data captured by Cisco's NetFlow provides high level fine grained data and application specific information at high speed while packet-level data captured in the form of pcap format by tools like tcpdump and ethereal are mostly used to debug applications and intercept network setup.

The main challenges in network measurement are the capability to identify and classify traffic accurately at high speed, timely export of the measurement data and the potential of storing huge amount of data. A methodology is discussed and reviewed on [17] that relies on the observation of the first five packets of a TCP connection to identify the application. A content-based classification approach to identify network applications is discussed in [18]. It is capable of accurately classifying and identifying traffic flows that could otherwise be classified incorrectly. References [19] and [20] discusses a Naïve Bayes estimator to categorize traffic by applications. It considers hand-classified network data that is given to a supervised Naïve Bayes estimator.

Rapid changes in the organisation and their environment are mostly through policy driven activities. The issue is addressed by using a policy repository and plug-in-modules of policy elements. The main problem is how to enforce policies no matter what kind of policies are required to be enforced and how do we provide flexibility and adaptability to this policy enforcement to accommodate future requirements? An integrated grid resource management architecture based on QoS-constraint policy discussed in [21] derives cost-sensitivity policy enforcement procedure which can be applied to this architecture in Grid over GMPLS (Generalize Multi-Protocol Label Switch) network. The article [22] discusses the Enter-

prise Policy Server, which is responsible for the installation and tracking of QoS policies on the network. This includes policy interpretation from high level definition into low level device configuration, policy conflict resolution, device communication, policy installation, and policy storage.

VI. Conclusions and Future Work

We have described the design and implementation of a flexible programmable passive network monitoring system. An approach to dynamically configure differentiated service capabilities in an open source software router platform is discussed. The proposed framework uses information collected from network monitoring, transport, middleware and application functions.

Initial results illustrate the performance of the proposed approach with respect to dynamic QoS behaviour of an open source software router, packet loss and time to re-configure. It shows that the packet loss and the time to re-configure the XORP+Click router according to the flow criteria set are negligible. Extensions of this specific theme of experiment would be to investigate the issues related to dynamic handling of out-of-profile traffic in relation to charging and pricing mechanisms.

In terms of the soundness of the approach, we can appeal to the basic Internet principles, in that we are not breaking the end-end model, nor do we demand that state be stored and refreshed in forwarding elements on behalf of end-to-end relationships. What we have provided is a set of information flows and a distributed processing framework that exploit the capabilities of a flow export function and demonstrate how the loop can be closed - i.e. as a case study of what flow export is for.

Although we have chosen a different specific platform, we would expect that the recommendations and RFCs of the IETF IPFIX WG would support the protocol functionality we need. However, protocol specifications are not written in an architectural vacuum, and we would draw the reader's attention to what we consider to be significant benefits of the CRANE architecture. And we must not forget that the strength of the IETF is that its recommendations can be ignored if a better approach is identified!

At this stage of development we are not in a position to answer obvious, or deeper, questions about the rationale for such a monitoring system and whether it is an asset to the IP community or another threat to the open Internet. Performance, security against all the usual threats, what is a flow anyway, is IP QoS any use: all these are still open. For now, we see its potential as a platform for discovering what is going on out there and responding to it.

References

[1] J. Quittek, T. Zseby, et al., *Requirements for IP Flow Information Export (IPFIX)*, RFC 3917, October 2004.

[2] Tarari Inc., *Regular Expression Agent Processor API Guide*, White paper, available from http://www.tarari.com/library.asp.

[3] J. Apisdorf, K. Claffy, et al., *OC3MON: flexible, affordable, high performance statistics collection*, In Proc. of the USENIX Tenth System Administration Conference (LISA X), Chicago, IL, September 1996.

[4] S. Waldbusser, R. Cole, et al., *Introduction to the Remote Monitoring (RMON) Family of MIB Modules*, RFC 3577, August 2003.

[5] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, RFC 3954, October 2004.

[6] G. Iannaccone, *CoMo: An open infrastructure for network monitoring - Research Agenda*, Intel Research Technical Report, February 2005.

[7] K. Zhang and E. Elkin, *XACCT's Common Reliable Accounting for Network Element (CRANE) Protocol Specification Version 1.0*, RFC 3423, November 2002.

[8] S. Leinen, *Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX)*, RFC 3955, October 2004.

[9] M. Handley, O. Hodson and E. Kohler, *XORP: An Open Platform for Network Research*, In Proc. of First Workshop on Hot Topics in Networks, October 2002.

[10] E. Kohler, R. Morris, et al., *The Click Modular Router*, ACM Trans. on Computer Systems, vol. 18, no. 3, August 2000.

[11] D. C. Schmidt and S. D. Huston, *C++ Network Programming: Systematic Reuse with ACE and Frameworks*, Addison-Wesley Longman, 2003, ISBN 0-201-79525-6.

[12] E.P. George Box, W.G. Hunter, J. Stuart Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building*, John Wiley & Sons, 1978.

[13] http://www.openldap.org/

[14] R. Rajan, et al., *A Policy Framework for Integrated and Differentiated Services in the Internet*, IEEE Network, vol. 13, no. 5, pp. 36-41, September/October 1999.

[15] D. Durham, J. Boyle, et al., *The COPS (Common Open Policy Service) Protocol*, RFC 2748, January 2000.

[16] http://dast.nlanr.net/Projects/Iperf/

[17] L. Bernaille, R. Teixeira, I. Akodkenou, et al., *Traffic classification on the fly*, ACM SIGCOMM Computer Communications Review, April 2006.

[18] A. W. Moore and D. Papagiannaki, *Towards the accurate identification of network application*, In Proc. of Passive and Active Measurement Conference, Boston, MA, March 2005.

[19] A. W. Moore and D. Zuev, *Internet traffic classification using Bayesian analysis techniques*, In Proc. of ACM SIGMETRICS, Banff, Alberta, Canada, June 2005.

[20] T. Auld, A. W. Moore, and S. F. Gull, *Bayesian neural networks for internet traffic classification*, IEEE Transactions on Neural Networks, vol. 18, no. 1, January 2007.

[21] H. Song, C. H. Youn, C. Han and C. Chen, *QoS-constraint Configuration Management Policy Enforcement Scheme in Grid over GMPLS Networks* , In Proc. of Internation Conference on Communication Technology, Guilin, China, November 2006.

[22] A. Melia, *Quality of service in enterprise networks* , IEE Colloquium on Services Over the Internet - What Does Quality Cost?, June 1999.